

# *Programming for Robotics*



**Newbury Park High School 2016**

# Basic Tele-Op using Templates

PFR.01.03

Created by LCpl Rice  
20160921

- VEX Joystick
- VEX Motors
- Types of Robot Driving
- Tele-Op programs
- Tank Drive Programming
- Arcade Programming

# Learning Objectives

Terminal = What you walk away with  
Enabling = What you will be evaluated on

## Terminal Learning Objectives

- Understand the differences between the different VEX motors
- Know how to create and use Tele-Op programs from scratch or from a template
- Know how to program a robot for Tank Drive or Arcade Drive

## Enabling Learning Objectives

- Understand Tank Drive, Arcade, and Holonomic drive modes in accordance with *What's Your Robot Drive Style*, Professor VEX, 2011.
- Understand how to create a Tele-Op program from scratch and use a Joystick to communicate with the robot according to xxx, xxx, 20xx.
- Be able to program Tank and Arcade drive modes using built in functions and from scratch using easyC, in accordance with XXX, xxx, 20xx.

# VEX Joystick

The VEX Joystick is the primary way for human control over a VEX robot using easyC or robotC.

They Joystick can be connected to the robot and/or a computer running easyC/robotC, and can be powered using AAA batteries or a rechargeable battery pack.

They Joystick will **NOT** communicate with the robot if the robot is using an autonomous program.



# VEX Joystick

You can connect the Joystick to the Cortex through using the Male-Male USB type A cable by plugging one end into the top of the cortex, and the other on the bottom of the Joystick.



# VEX Joystick

Take the RJ12 serial cable (top right) and plug it into the *Programming* jack in the Joystick (bottom right)

And the other side into the serial to usb connector (left) with the USB cable plugged into your computer running easyC/robotC.



# VEX Joystick

Optionally, you can also connect a Partner Joystick or a second regular Joystick for dual joystick control by connecting the two via an RJ12 cable in each controller's *Partner* port.

We will not be doing that in this lesson.

Partner Joystick can be identified by the dark grey VEX cover, and **NO USB PORT** on the bottom.



Controller 1

Controller 2

# Joystick Layout

- A. Partner RJ11 Jack
- B. Program RJ12 Jack
- C. On/Off Power Switch
- D. Ethernet Competition Jack RJ45
- E. Top Right Bumper,  
Digital Channel 6, Button 2
- F. Bottom Right Bumper,  
Digital Channel 6, Button 1
- G. Top Left Bumper,  
Digital Channel 5, Button 2
- H. Bottom Left Bumper,  
Digital Channel 5, Button 1





# Joystick Layout

- A. Left **Analog** Stick
  - Up/Down Channel 3
  - Left/Right Channel 4
- B. Indicator Lights
- C. Right **Analog** Stick
  - Up/Down Channel 2
  - Left/Right Channel 1
- D. Digital Channel 7
  - 1 = Down
  - 2 = Up
  - 3 = Left
  - 4 = Right
- E. Digital Channel 8
  - 1 = Down
  - 2 = Up
  - 3 = Left
  - 4 = Right



# VEXnet

We will not be uploading code to the robot over VEXnet due to it having an unstable connection.

The project at the end of the class will require VEXnet to drive the robot, but not program.

Refer to PFR.01.01 for how to connect over VEXnet.

# VEX Motors

3-Wire Motor  
3-Wire Servo  
269 Motor  
269 Motor w/ Encoder  
393 Motor  
393 Motor (High Speed)  
393 Motor (Turbo)  
393 Motor w/ Encoder  
393 Motor (High Speed) w/ Encoder  
393 Motor (Turbo) w/ Encoder  
Motor Controller 29

393



269 w/ Encoder



393 w/ Encoder



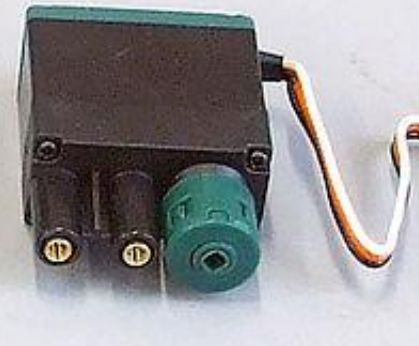
Motor Controller



269



3-Wire Motor



3-Wire Servo

# VEX Motors

The 3 wire motor and servo can both be plugged in any 3 wire motor port on the cortex, but any 269 or 393 motor requires either port 1 or 10 (the 2 wire ports) or must be plugged into a motor controller that is plugged into any other motor port.

The 3 wire motor and servo, as well as the 269 motor are the same size, while the 393 motor is slightly larger.

Note: High Speed and Turbo geared motors are visibly identical to their normal counterparts.

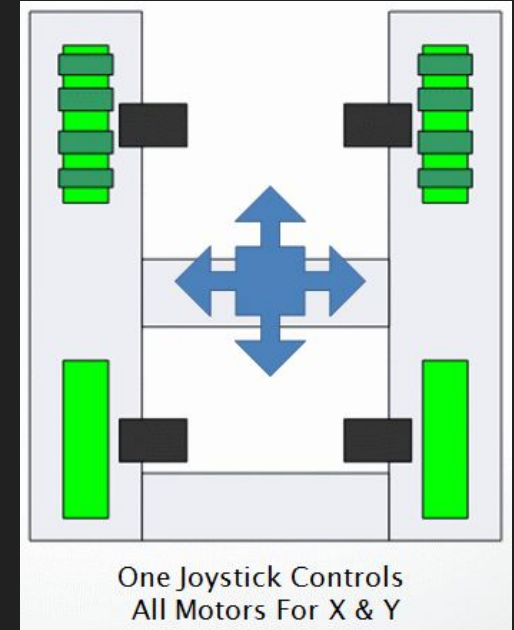
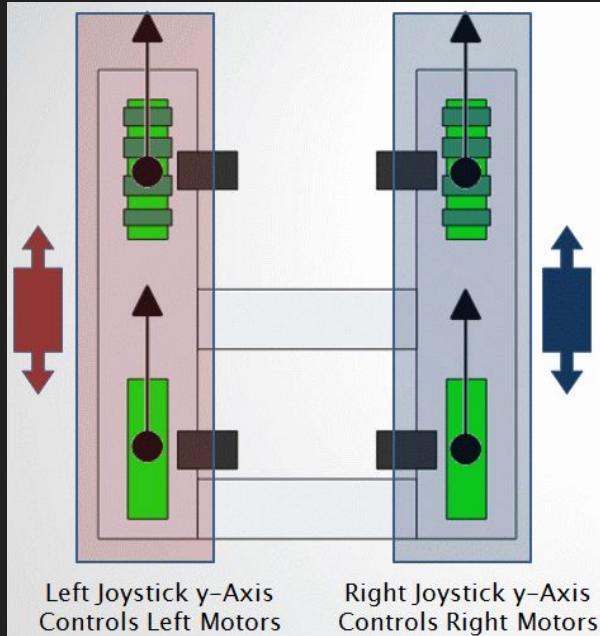
The integrated encoders on the top of any 269 or 393 motor add some size to the top of each motor, and must be wired together in a bus. More detail will be covered on this topic in a later chapter.

Controller Configuration has a section for identifying if a motor has an encoder or not:

Motor Type Information
n/a - Motor Type is not provided
Standard - Motor Module without Integrated Encoder
Small IME - 269 with Integrated Encoder
Big IME - 393 with Integrated Encoder
Big IME HS - 393 High Speed Gearing with Integrated Encoder

# Modes of Robot Driving

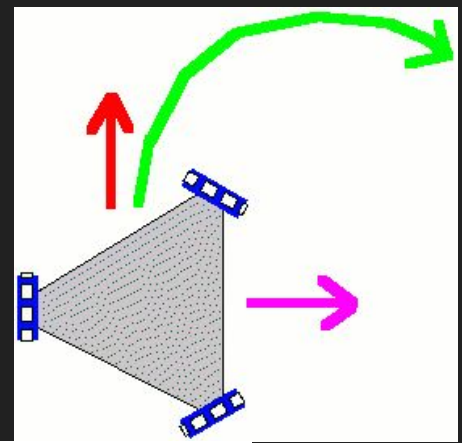
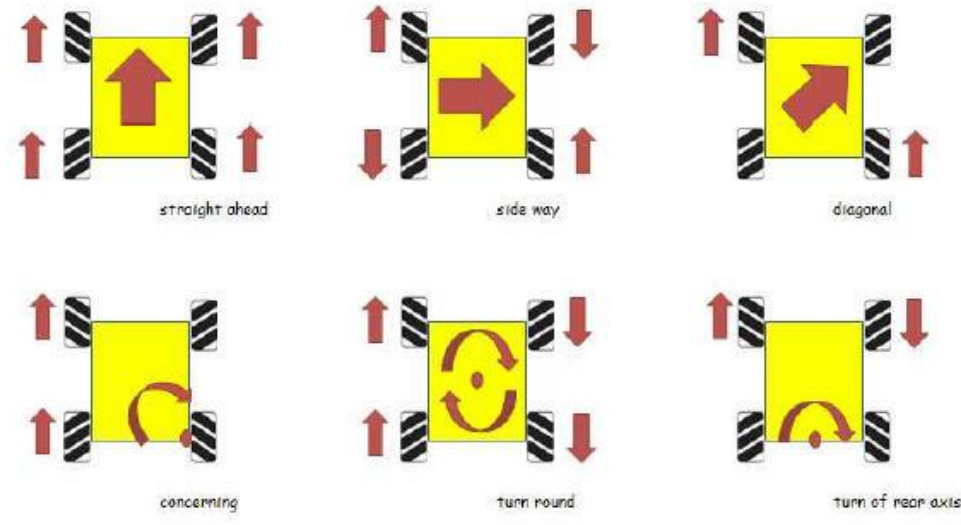
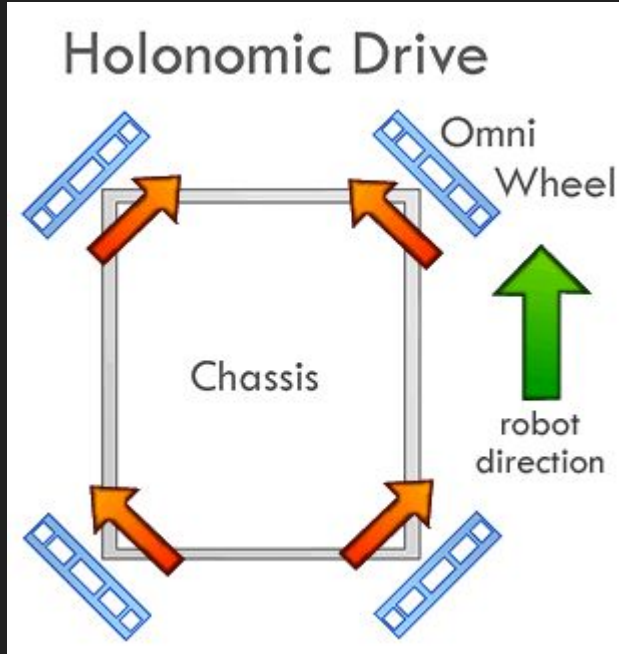
- Tank Drive (Requires FWD/BCK for each side)
- Arcade (Requires FWD/BCK and Rotate)



# Modes of Robot Driving

- Holonomic (Requires FWD/BCK, Rotate, and Strafe)

(Left & Right) Omni Wheels  
(Bottom) Mecanum

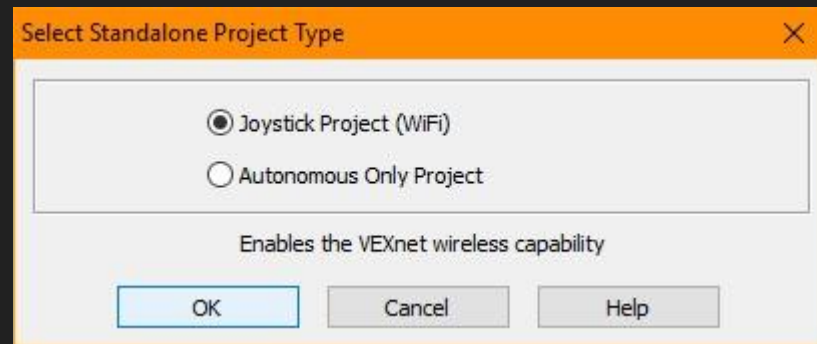
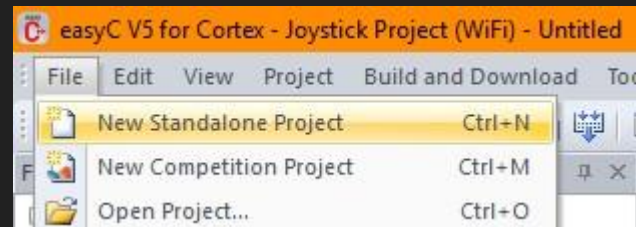


# New Tele-Op Program

Tele-Op stands for Teleoperation and it indicates operation of a machine at a distance. More commonly referred to as Remote Control.

For our usage with easyC and VEX, a **Tele-Op program REQUIRES the VEX Joystick** to be communicating with the robot in order to run the program.

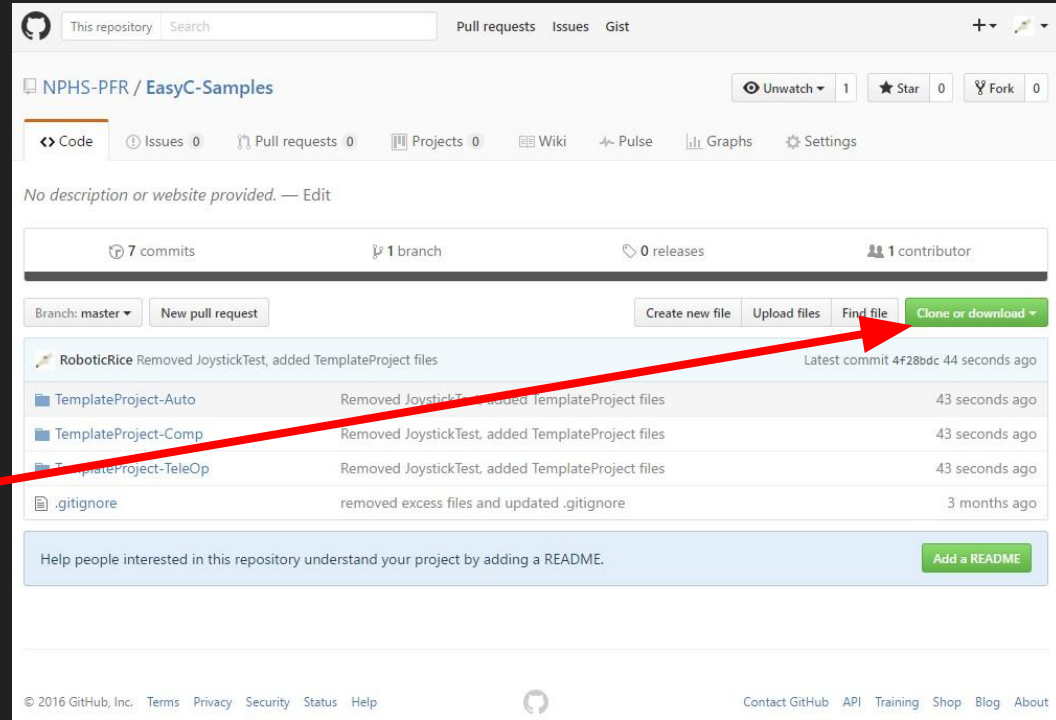
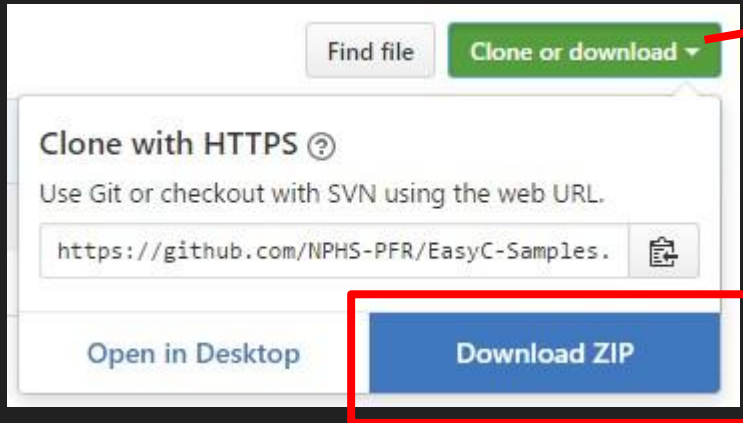
If the Joystick is either not connected (via cable or VEXnet) or has no battery when the cortex is first turned on, the robot code will not start until the Joystick is connected **and** powered.



# EasyC Templates

All sample programs for this course can be found hosted on gitHub's NPBS-PFR group under "EasyC-Samples"

<https://github.com/NPBS-PFR/EasyC-Samples>

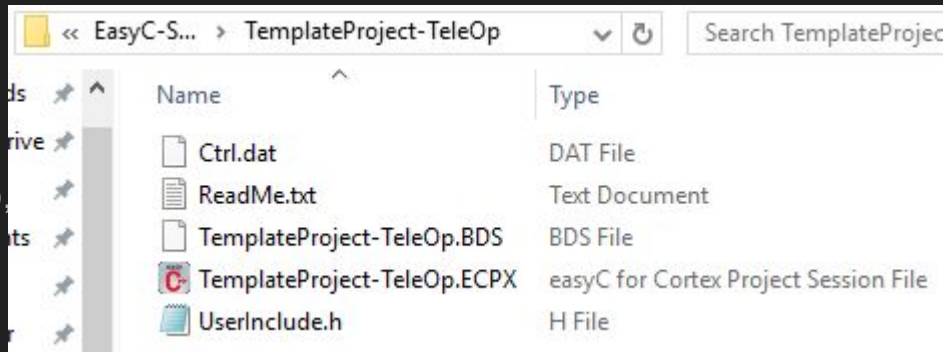




# EasyC Templates

The templates are organized by project type; TeleOp, Autonomous, and Competition.

Inside each project folder contains the files pictured.



.ECPX is the Project file that you open with easyC.

.BDS and UserInclude.h are both files you should never mess with.

ReadMe.txt is autogenerated, and is optional to edit. It has no effect on the actual program.

Ctrl.dat contains the Controller Configuration data, and can be moved to another project folder to transfer all your Motors, Analog, and Digital I/O ports you pre-configured.

For every project you create in this class or the robotics club, please name it as follows:

**FirstnameLastinitial-ProjectName-ProjectType**

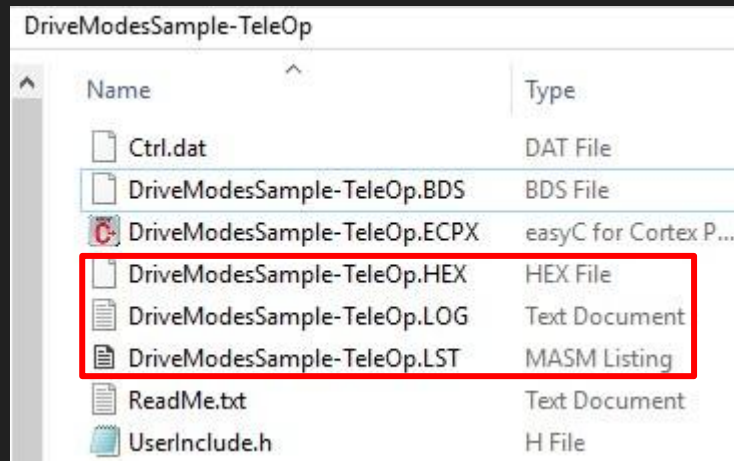
*Example: SamR-Sample-Auto*

# EasyC Templates

## NOTE:

.HEX .LOG and .LST files are created when you compile the program, and are not needed for the program to run, but instead are used for easyC's built in troubleshooter and debugger.

Your project may contain these files, the but they are set to be ignored by gitHub, and should not be present in any sample programs you download.



Name	Type
Ctrl.dat	DAT File
DriveModesSample-TeleOp.BDS	BDS File
DriveModesSample-TeleOp.ECPX	easyC for Cortex P...
DriveModesSample-TeleOp.HEX	HEX File
DriveModesSample-TeleOp.LOG	Text Document
DriveModesSample-TeleOp.LST	MASM Listing
ReadMe.txt	Text Document
UserInclude.h	H File

# Manual Tank Drive

Logically, what do we need to do in order to drive the robot?

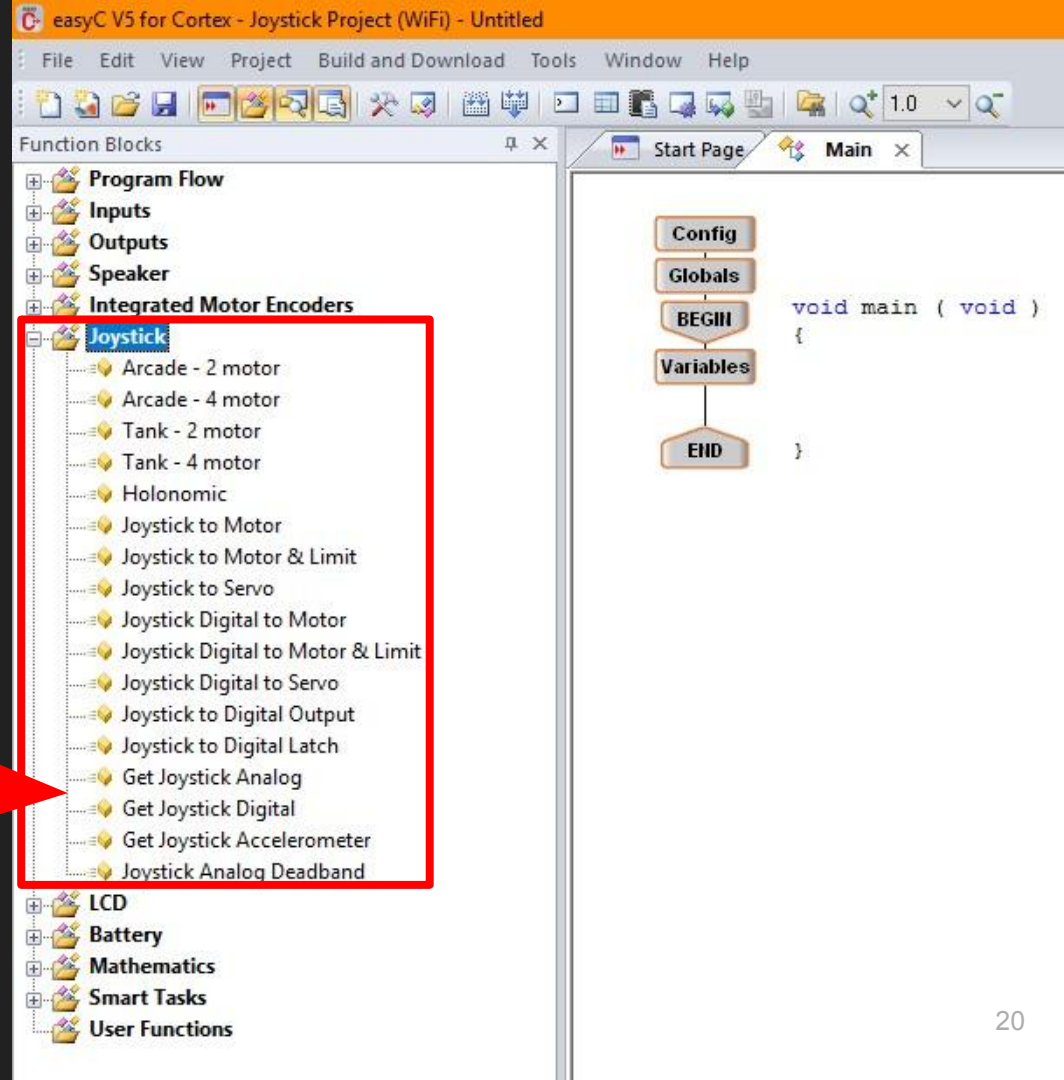
- Get value from Joystick
- Set motor speed to that value
- LOOP!

# Manual Tank Drive (Get Joystick Value)

All Joystick commands are located in the Function Block “*Joystick*”

EasyC provides several preconfigured functions for driving, but we are going to start with the basics:

**Get Joystick Analog**  
**Get Joystick Digital**



# Manual Tank Drive (Get Joystick Value)

Since the **Analog** sticks are *analog*, we use the function: *Get Joystick Analog*.

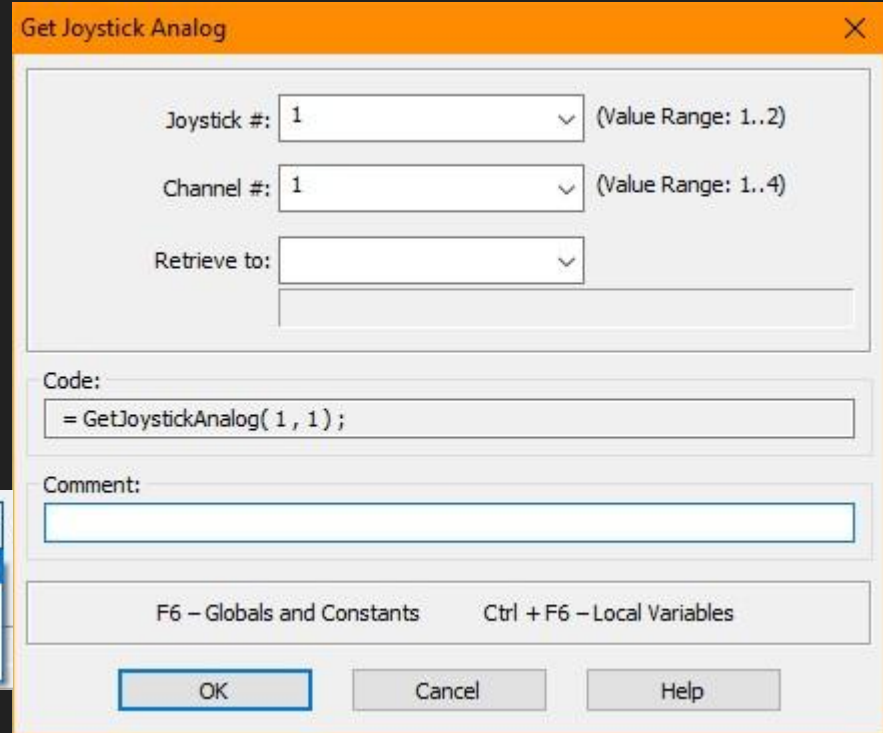
Joystick: The joystick are we using, 1 being the one directly connected to the cortex, and 2 being the optional partner connected to joystick 1.

Channel: 1 through 4, corresponding to the Up/Down or Left/Right of left and right Analog sticks.



Channel #: 1

Retrieve to: 1, 2, 3, 4



Get Joystick Analog

Joystick #: 1 (Value Range: 1..2)

Channel #: 1 (Value Range: 1..4)

Retrieve to:

Code:

```
= GetJoystickAnalog( 1 , 1 );
```

Comment:

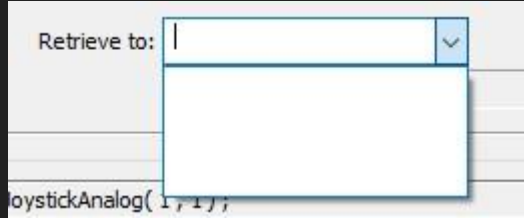
F6 - Globals and Constants      Ctrl + F6 - Local Variables

OK Cancel Help

# Manual Tank Drive (Get Joystick Value)

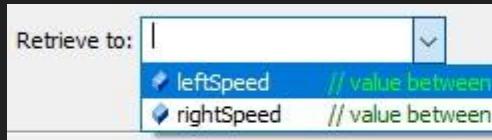
Retrieve to:

This is a variable that will be set to the value of the joystick we selected. The value will be an integer between -127 (all the way down), and 127 (all the way up), where 0 is centered.

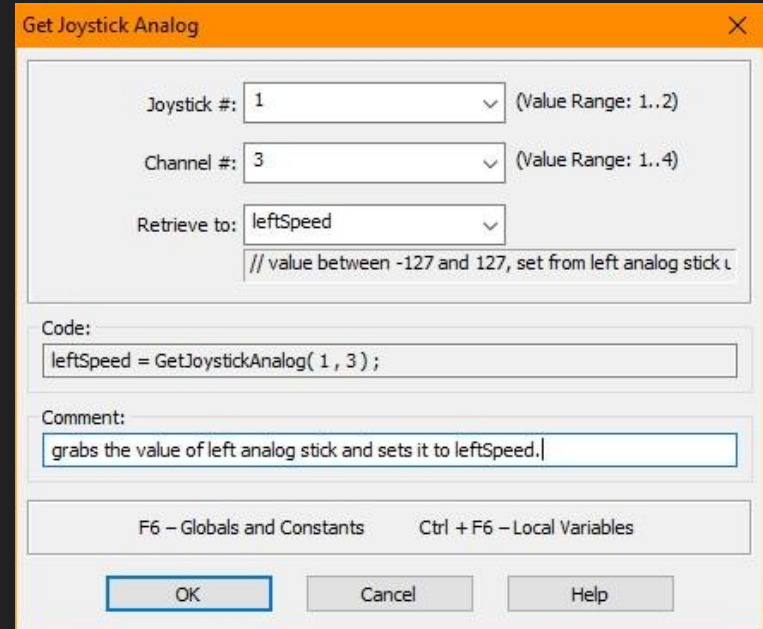


## Local Variables

#	Type	Name	Value
1	int	leftSpeed	0
»			



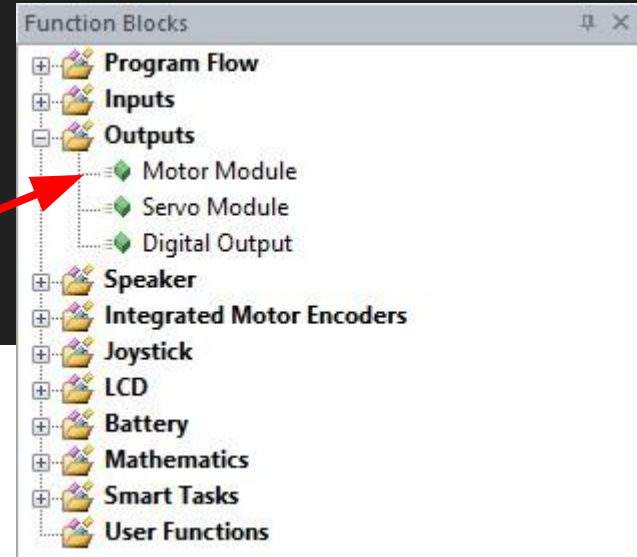
Create a local variable for the leftSpeed and another for rightSpeed. Both variables should be of type int and value of 0.



# Manual Tank Drive (Set Motor Speed)

Now that we have gotten the value of both analog sticks, we need to set the motor to that value. Motors are considered *Outputs* and can be found under that Function Block category.

Drag Motor Module to the end of the program.



```
void main ( void )  
{
```

```
leftSpeed = GetJoystickAnalog( 1 , 3 ) ; // grabs the value of left analog stick and sets it to leftSpeed.
```

```
rightSpeed = GetJoystickAnalog( 1 , 2 ) ; // grabs the value of right analog stick and sets it to rightSpeed.
```

```
}
```

# Manual Tank Drive (Set Motor Speed)

If you configured your controller correctly, the drop down box for motor number should tell you what each motor port is.



Motor Direction: You can select any of the three presets, or User Value. User value can be a value you type into it, or you can select a variable from the drop down box.



**Motor Module**

Motor Number:  (Value Range: 1..10)

Motor Direction (Value Range: -127..127):

Counter-clockwise ☐

Stop ☐

Clockwise ☐

User Value ☒

Code:

Comment:

F6 – Globals and Constants      Ctrl + F6 – Local Variables

A diagram of a motor module with two cylindrical motors. A red arrow points from the left motor to the right motor, indicating a clockwise direction. The left motor is labeled 'CCW +127' and the right motor is labeled 'CW -127'.



# Manual Tank Drive (Set Motor Speed)

Pop Quiz:

What's wrong with our code as of right now?

One of the motors on our robot is physically inverted, and therefore requires the inverse of its speed. You can add  $*-1$  to whichever motor requires the change.



```
leftSpeed = GetJoystickAnalog( 1 , 3 ) ; // grabs the value of left analog stick and sets it to leftSpeed.
```



```
rightSpeed = GetJoystickAnalog( 1 , 2 ) ; // grabs the value of right analog stick and sets it to rightSpeed.
```



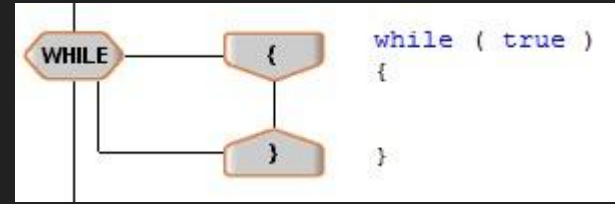
```
SetMotor ( 2 , leftSpeed ) ; // take the left motor and set it to the left speed
```



```
SetMotor ( 3 , rightSpeed ) ; // take the right motor and set it to the right speed
```

# Manual Tank Drive (Loop)

Since our loop does not need to stop, let's create an infinite loop. This can be done as `While(true)`, `While(1==1)`, `While(1)`, or any other condition that will always be true.



The code that will be looped needs to be located between the `{` and `}` blocks.

While Loop

Expression:

while (  )

Add Variable:  Add Operator:

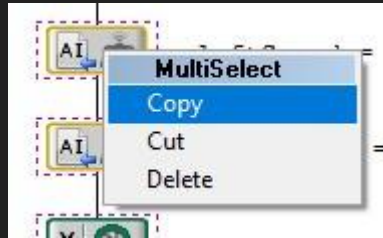
Code:

Comment:

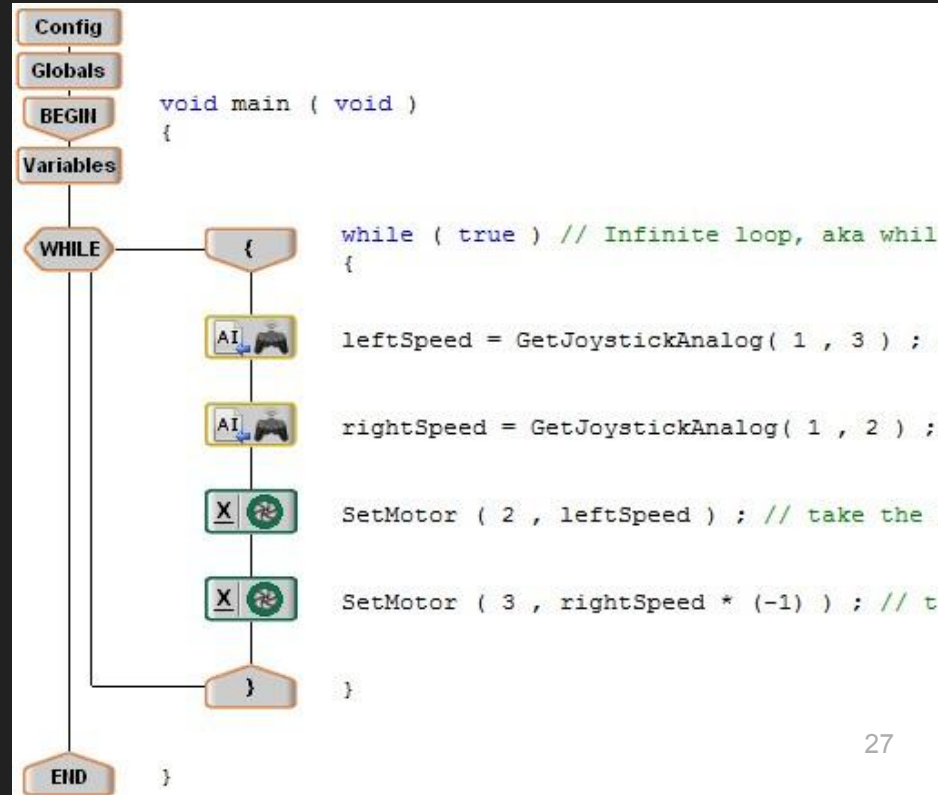
F6 – Globals and Constants    Ctrl + F6 – Local Variables

# Manual Tank Drive (Loop)

Shift-Click each of the code blocks to select multiple blocks. Selected blocks have a purple dashed outline on them.

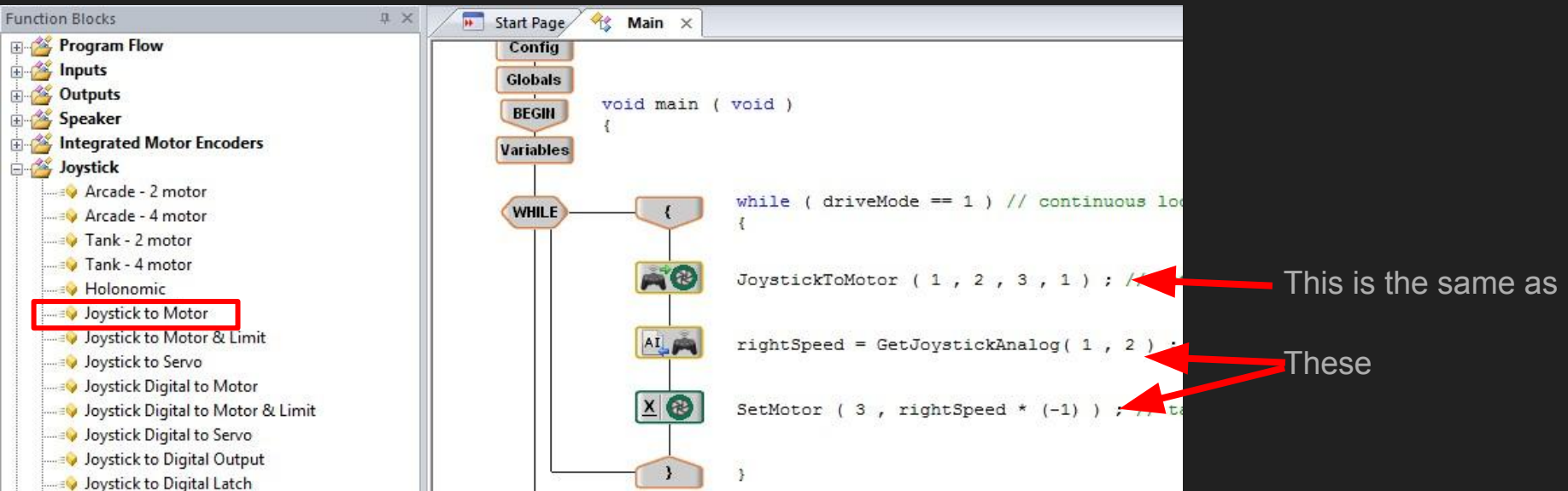


Right click on the selected blocks to copy/cut them, and right click in the while loop to paste.



# Condensed Tank Drive

EasyC provides some functions that programmers use often that make life easier. One such function is the *Joystick To Motor* function, which combines the *Get Joystick Analog* and *Set Motor* functions into one.



The screenshot displays the EasyC software interface. On the left, the 'Function Blocks' panel lists various blocks under the 'Joystick' category. The 'Joystick to Motor' block is highlighted with a red rectangle. The main workspace shows a 'Main' window with a C++ code editor. The code is as follows:

```
void main ( void )  
{  
    while ( driveMode == 1 ) // continuous loop  
    {  
        JoystickToMotor ( 1 , 2 , 3 , 1 ) ;  
        rightSpeed = GetJoystickAnalog( 1 , 2 ) ;  
        SetMotor ( 3 , rightSpeed * (-1) ) ;  
    }  
}
```

Red arrows point from the text 'This is the same as' to the `JoystickToMotor` function call in the code. Another set of red arrows points from the text 'These' to the `GetJoystickAnalog` and `SetMotor` function calls, indicating that the `JoystickToMotor` block encapsulates these two operations.

# Condensed Tank Drive

The *Joystick To Motor* function window has many of the same fields the previous two windows used, just in one, except with the addition of the *Invert Direction* field.

Invert direction is 0 for false (do not invert)  
Or 1 for true (invert direction).

If the direction is inverted, whatever value you set the motor to go, it will instead go:

value \* -1

Joystick to Motor

Transmitter:  
Joystick #: 1 (Value Range: 1..2)

Transmitter Channel:  
Channel: 2 (Value Range: 1..4)

Motor Number (Value Range: 1..10) and Invert Direction (Value Range: 0 - default, 1 - invert):

Motor:  
3  
// rDrive

Invert Direction:  
1

Invert Direction:

1
0 1 0
0 1 1

Code:  
JoystickToMotor ( 1 , 2 , 3 , 1 );

Comment:  
Get the value of right analog stick up/down and set it to right motor (which is inverted)

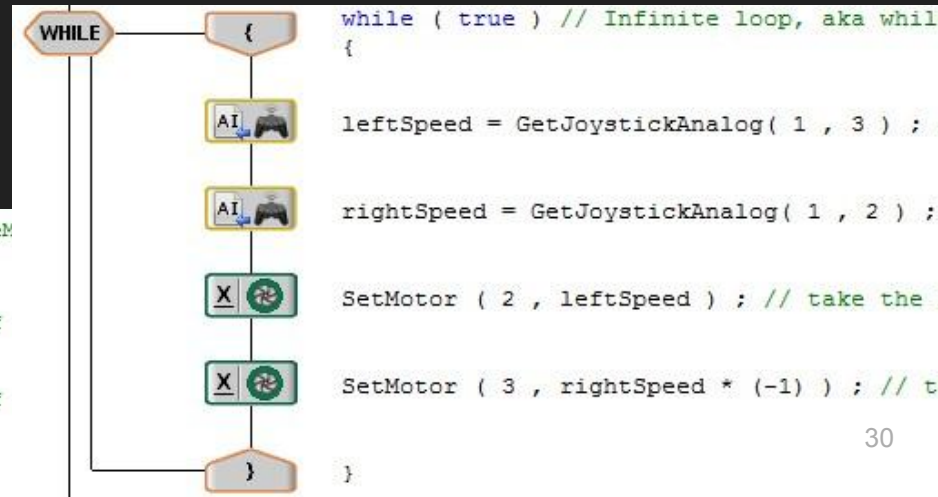
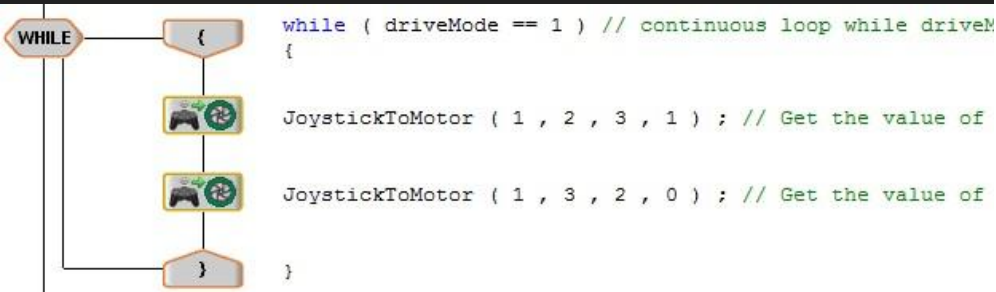
F6 – Globals and Constants      Ctrl + F6 – Local Variables

OK Cancel Help

# Condensed Tank Drive

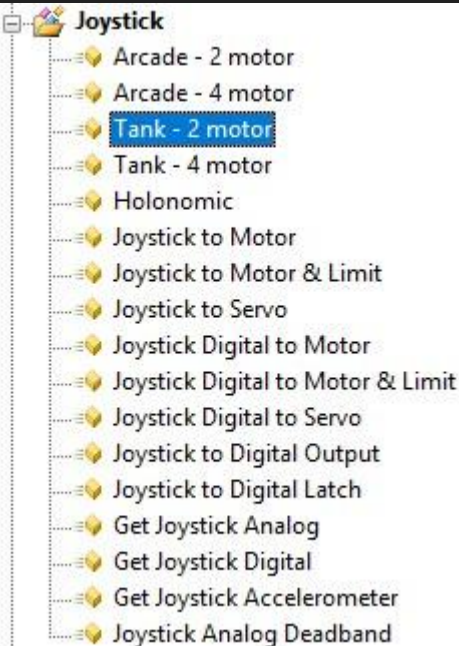
As you can see, the condensed version is shorter, and does **not** require any variables, which means it's significantly more efficient.

But EasyC takes this one step further...



# Built In Tank Drive

The Tank drive built-in function does everything in one line, and can work for 2 or 4 motor robots. The function can be found under the



Joystick function block category.

The function window has all of the same fields as before, just combined into one window.

Tank - 2 motor

Transmitter: Joystick #: 1 (Value Range: 1..2)

Transmitter Channel: Left Channel: 3 (Value Range: 1..4) Right Channel: 2 (Value Range: 1..4)

Motor Number (Value Range: 1..10) and Invert Direction (Value Range: 0 - default, 1 - invert):

Left Motor: 2 //lDrive Invert Direction: 0

Right Motor: 3 //rDrive Invert Direction: 1

Code: Tank2 ( 1 , 3 , 2 , 2 , 3 , 0 , 1 );

Comment: Takes left analog stick and sets it to left motor, AND right alaog stick and sets it to right motor inverted

F6 – Globals and Constants Ctrl + F6 – Local Variables

OK Cancel Help

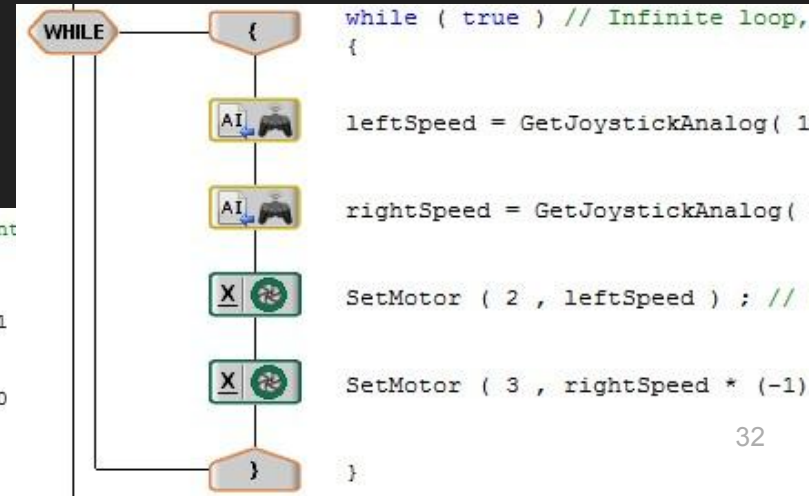
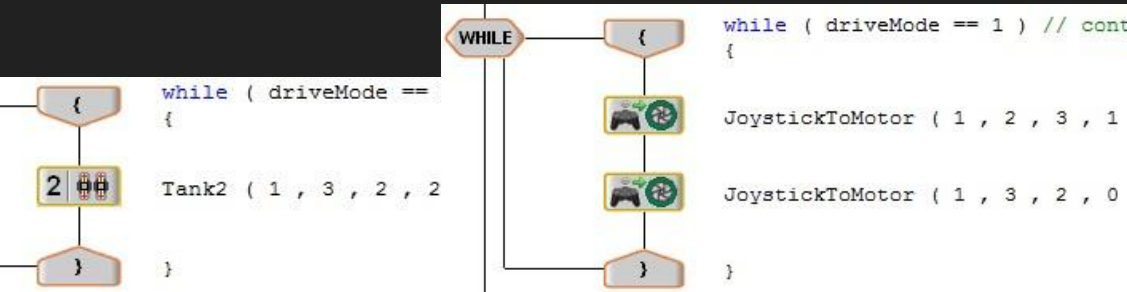
# Built In Tank Drive

The built in function is both the easier and the most efficient version of tank drive, but it does have one drawback, in that you have less control over what's actually happening.

Can anyone think of an example that would be difficult or impossible with the built in function?

Example:

Throttle maximum speed to 75% capacity.

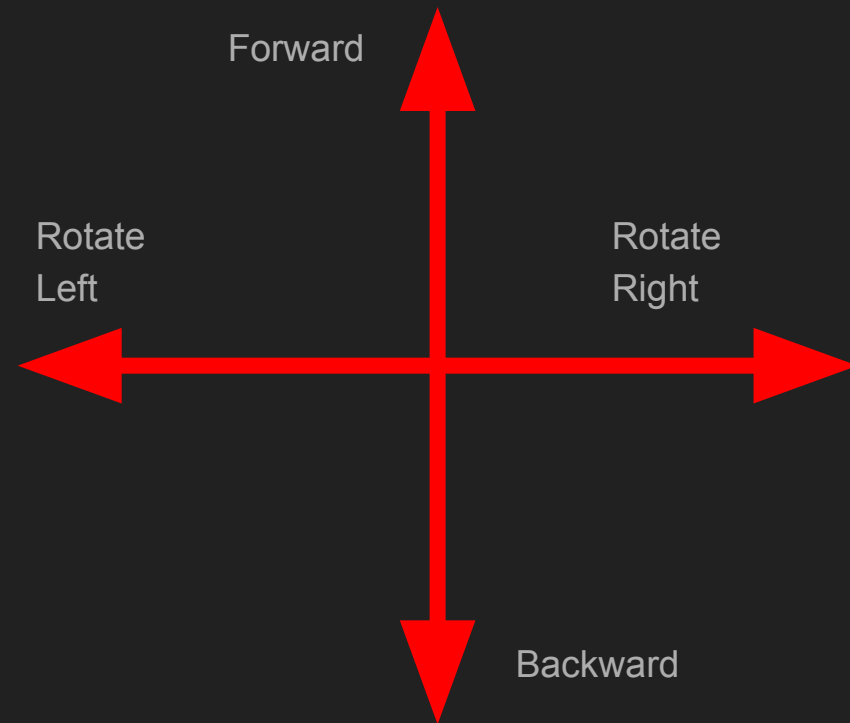




# Traditional Arcade Drive

Similar to the way we learned about Tank Drive, we will first manually program the Arcade drive mode, then use the built-in function.

Arcade drive works slightly different than tank drive, in that some arithmetic must be done in order for the robot to drive correctly.



# Traditional Arcade Drive

If we change the Get Joystick Analog to:

```
AI [Joystick Icon] fwdSpeed = GetJoystickAnalog( 1 , 3 ) ; //  
AI [Joystick Icon] rotateSpeed = GetJoystickAnalog( 1 , 2 ) ;
```

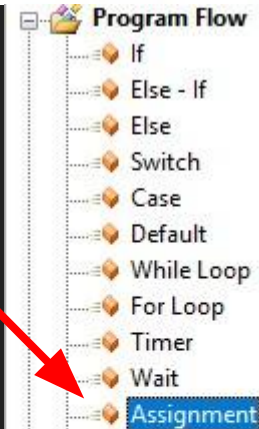
Then, both motors needs to move forward (or backward if negative) the value of fwdSpeed, which is between -127 and 127.

BUT - How do we rotate if both motors are going the same speed?

**Add** rotate speed to one side, and **subtract** it from the other side, that way if the value of rotate is 50, left speed will be 50 faster, and right speed will be 50 slower.

```
x=1+2 leftSpeed = fwdSpeed + rotateSpeed ;  
x=1+2 rightSpeed = fwdSpeed - rotateSpeed ;
```

\*The above block is the Assignment code block.



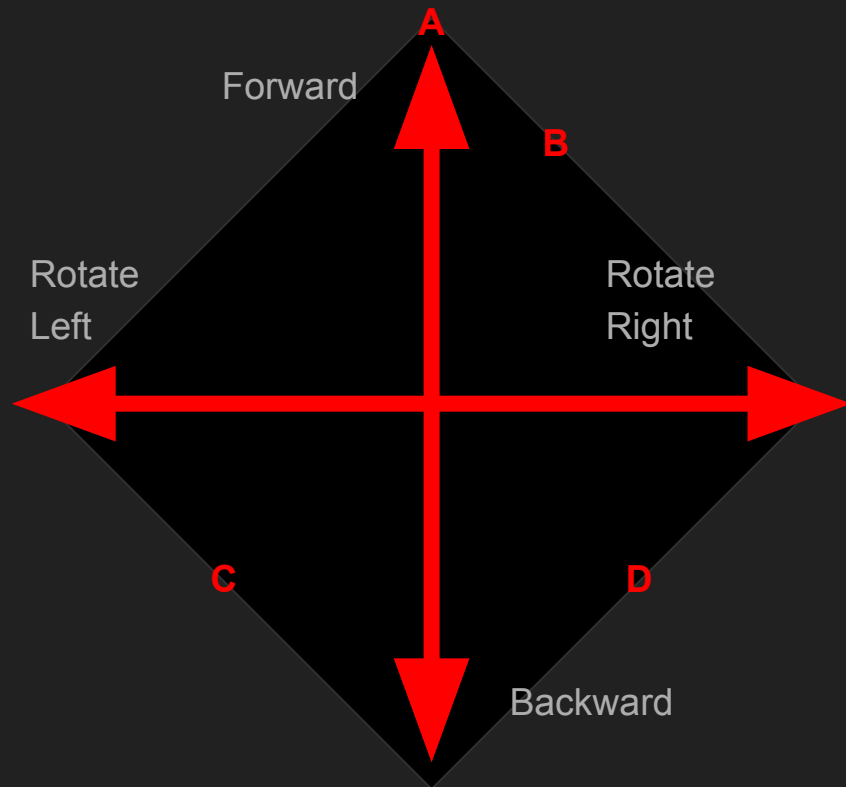
# Traditional Arcade Drive

But wait! If we have fwdSpeed plus rotateSpeed, can't we get a value of 254 (127+127), which is out of the range of the motor's speed?

Actually, no. That's because of the physical limitations on the analog stick, see examples:

- A: fwdSpeed = 127, rotateSpeed = 0
- B: fwdSpeed = 100, rotateSpeed = 27
- C: fwdSpeed = -63, rotateSpeed = -63
- D: fwdSpeed = -63, rotateSpeed = 63

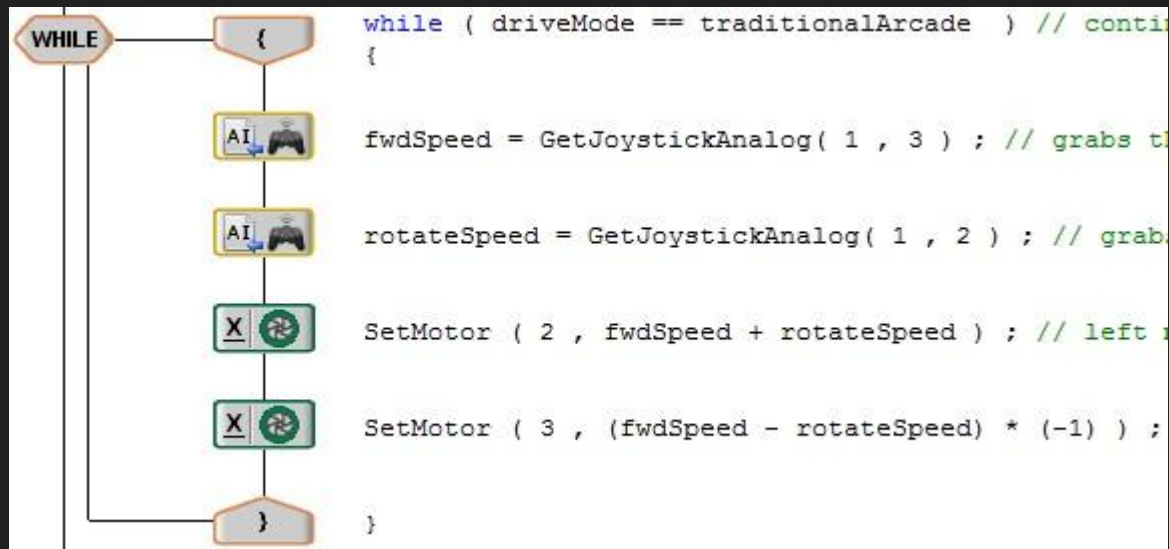
What if we used up/down from one joystick, and rotate from another?



# Traditional Arcade Drive

Reducing the code to the simplest form (by doing the arithmetic in the *Set Motor* function rather than in a separate *Assignment* function) results in the following:

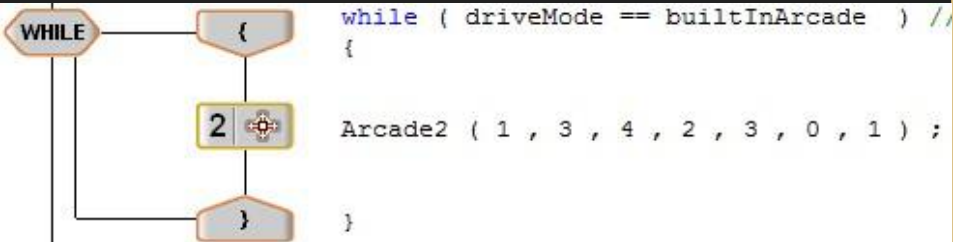
Remember: the  $\ast (-1)$  is to invert one motor who is physically flipped.



# Built In Arcade Drive



Arcade is a built in function that will reduce everything we just did to one line, similar to what the Tank function did.



```
while ( driveMode == builtInArcade )  
{  
  Arcade2 ( 1 , 3 , 4 , 2 , 3 , 0 , 1 ) ;  
}
```

### Arcade - 2 motor

Transmitter:

Joystick #: 1 (Value Range: 1..2)

Transmitter Channel:

Forward/Reverse Channel: 3 (Value Range: 1..4)

Rotate Channel: 4 (Value Range: 1..4)

Motor Number (Value Range: 1..10) and Invert Direction (Value Range: 0 - default, 1 - invert):

Left Motor	Right Motor
2	3
// lDrive	// rDrive
Invert Direction	Invert Direction
0	1

Code:

```
Arcade2 ( 1 , 3 , 4 , 2 , 3 , 0 , 1 ) ;
```

Comment:

```
fwd/bck on left analog stick moves left and right motor fwd/bck, while left/right on left analog stick rotates the robot
```

F6 - Globals and Constants    Ctrl + F6 - Local Variables

OK    Cancel    Help

# Lab

Create a program that can use Arcade Drive for either analog stick, simultaneously. This means, the driver (you) can drive the robot using just the right analog stick, just the left analog stick, or both without needing to press any buttons to switch modes. I expect to see comments! & wait for LCD button press.

## CAUTION:

Do NOT set the value of either motor to be greater than 127 or less than -127.

## Extra-Credit\*:

The robot must drive at the same speed using the original arcade function as it does with your dual arcade function.

\*There's actually no extra credit in this class...

fin

# Index

- Title Page
- Learning Objectives
- VEX Joystick
- Joystick Layout
- VEXnet
- Modes of Robot Driving
- New Tele-Op Program
- EasyC Templates
- Manual Tank Drive
  - Get Joystick Value
  - Set Motor Speed
  - Loop
- Condensed Tank Drive
- Built In Tank Drive
- Traditional Arcade Drive
- Built In Arcade Drive
- Lab